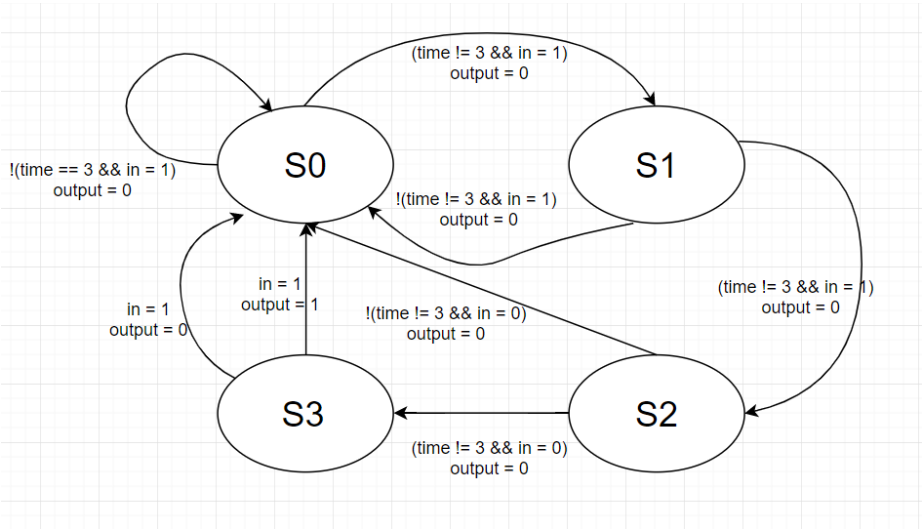
Lab 4-1 Mealy sequence detector

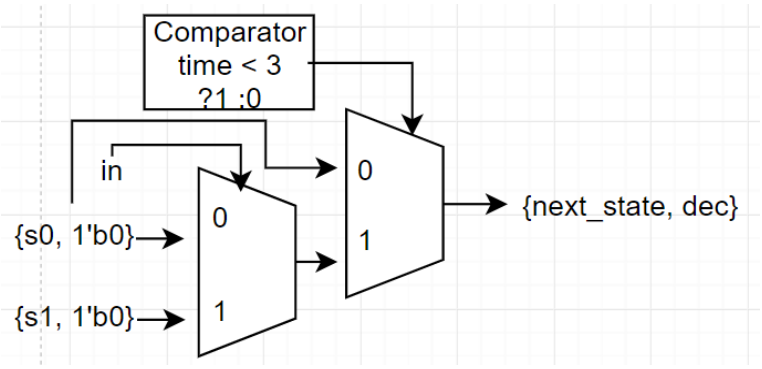
State transition

首先先定義好在每一個 state 時，input 及 time 的值會對應到何種 next_state 及 output，time 是一個 register，若是 time == 3，代表已經數到第四個數，接下來要強制返回 S0。以下為 state transition diagram。

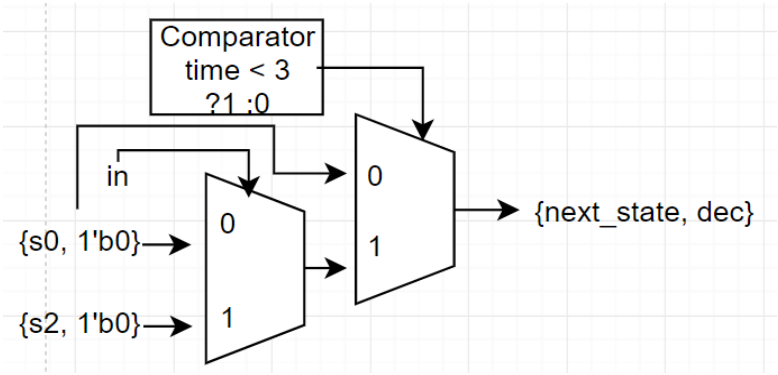


Design logic circuit

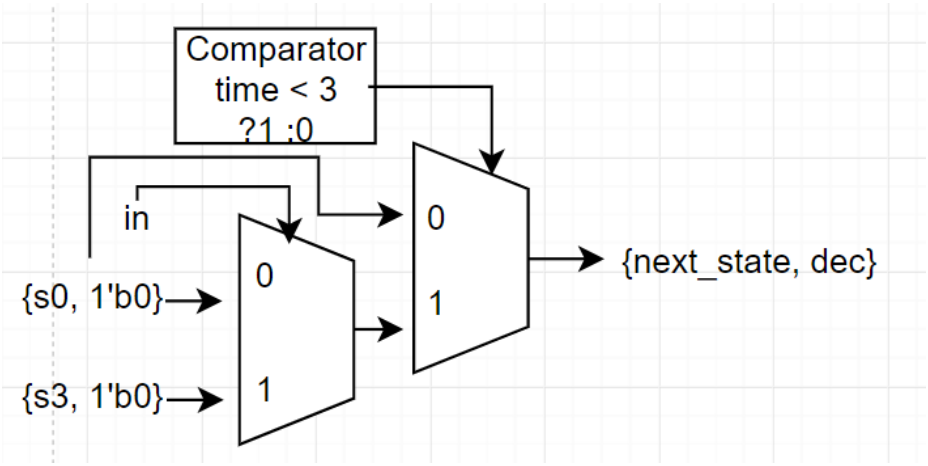
在 S0 時，若偵測 time == 3，強制返回 S0，若不等於 3 則繼續判斷，若 in 為 0，同樣留在 S0，若 in 為 1，轉換到 S1。以下為 logic circuit



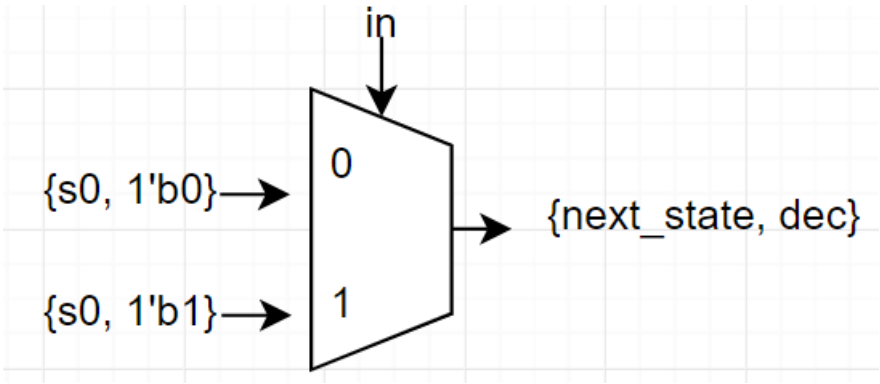
在 S1 時，若偵測 time == 3，強制返回 S0，若不等於 3 則繼續判斷，若 in 為 0，同樣留在 S0，若 in 為 1，轉換到 S1。以下為 logic circuit



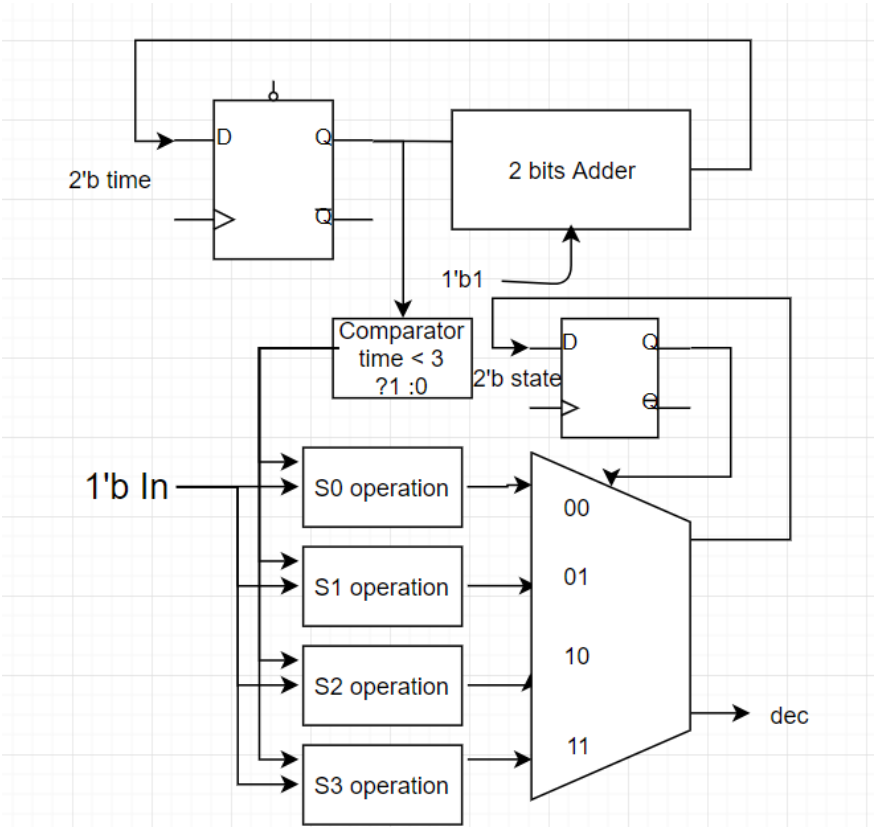
在 S2 時，若偵測 time == 3，強制返回 S0，若不等於 3 則繼續判斷，若 in 為 0，同樣留在 S0，若 in 為 1，轉換到 S1。以下為 logic circuit



在 S3 時，若 in 為 0，output 為 0，若 in 為 1，output 為 1，next_state 接轉換到 S0。以下為 logic circuit



以下為完整邏輯電路

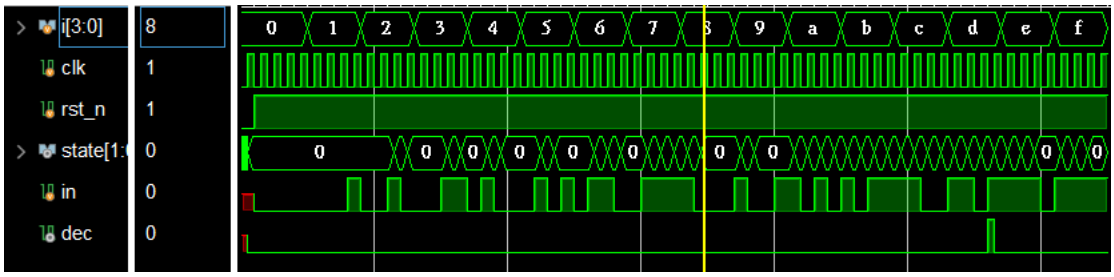


Verification

利用 verilog 驗證邏輯電路是否為正確

Testbench 設計方式

將每組 4bit 的組合都跑過一次，驗證是否只有在 input = 4'b1101 時 output 會等於 1，且若 1101 連續但 1101 在不同組的數字上(如 : 00110100)，結果要為錯誤。

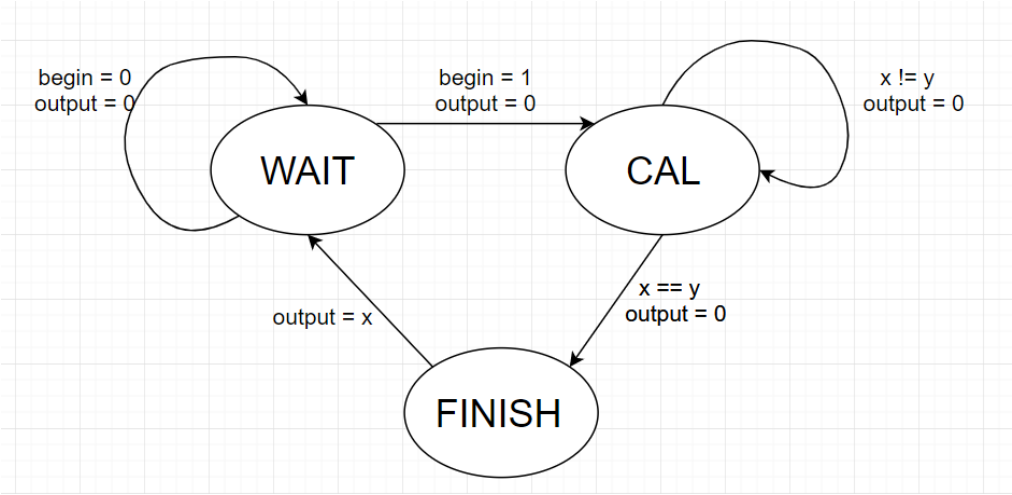


結果為所預期之結果，只有在輸入為 1101 時解果是正確

Lab 4_2 Greatest Common Divider

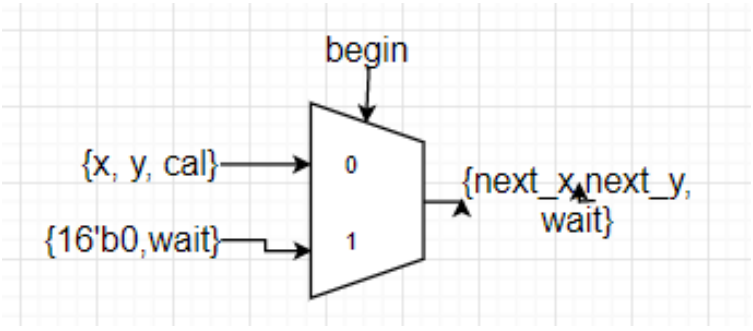
State Transition

首先先將 state diagram 做出來，定義在每個 state 的 signal 會對應到什麼樣的 output。

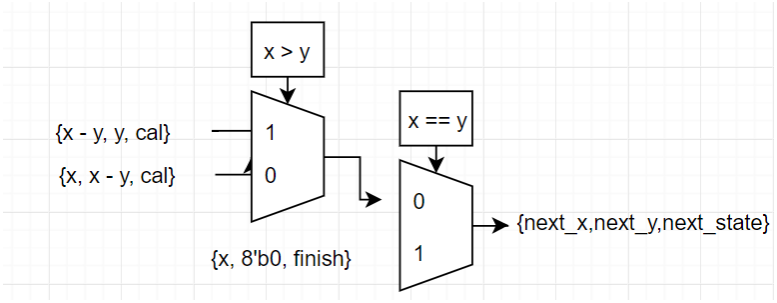


Design logic circuit

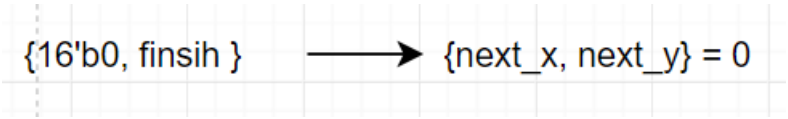
在 state WAIT 時，若偵測到 input begin 為 1，會進入下一個 state(CAL)，否則會停留在 begin 階段，如下圖()所示。



在 state CAL 時會經過兩個多功器，若 reg x, y 的值相等，將 x, y 的值保留，直接進入下一個 state finish；若 x, y 的值不相等，另一個多功器會做減法運算，減法運算會將較大的值保留，較小的數值留下，而 state 一樣停留在 cal 階段。

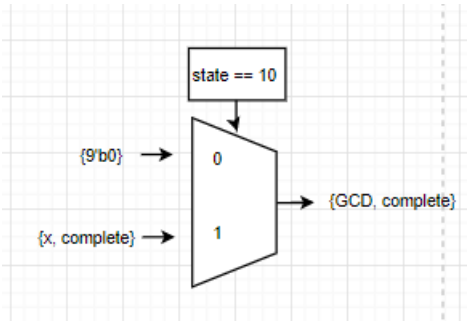


在 finish state 時直接將 next_state 設為 wait_state，x, y 的值都設為 0

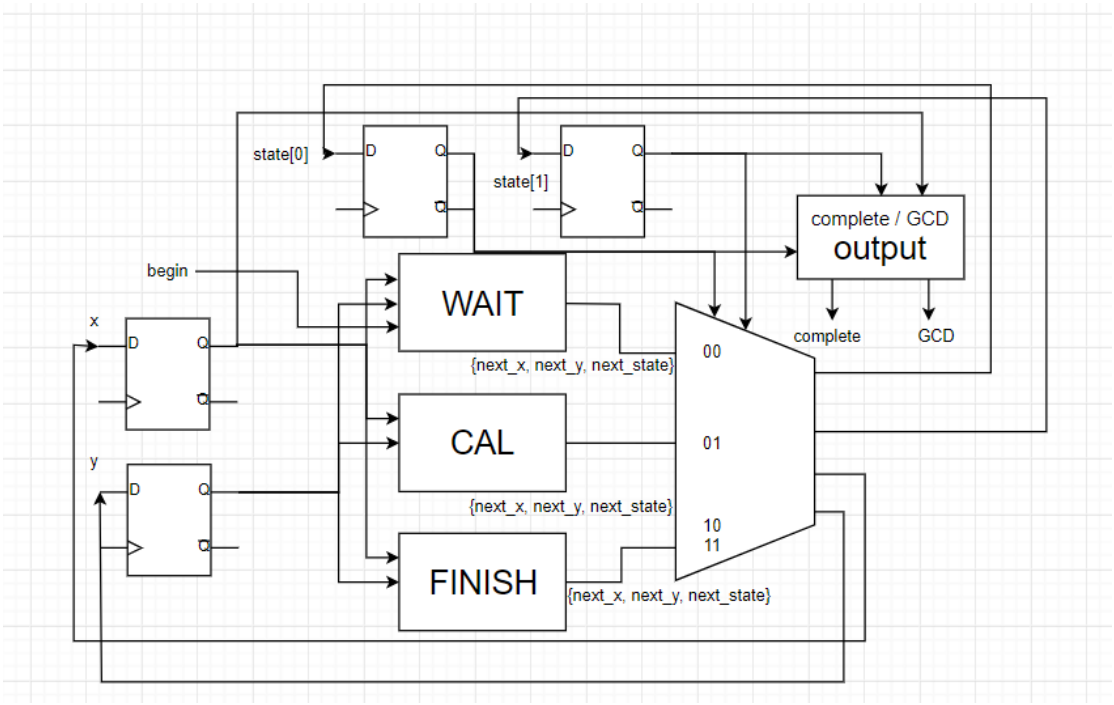


Output Assignment

若 state 為 finish state，將運算好的 GCD 值輸出，complete 值設為 1，若否，output 值全為 0。



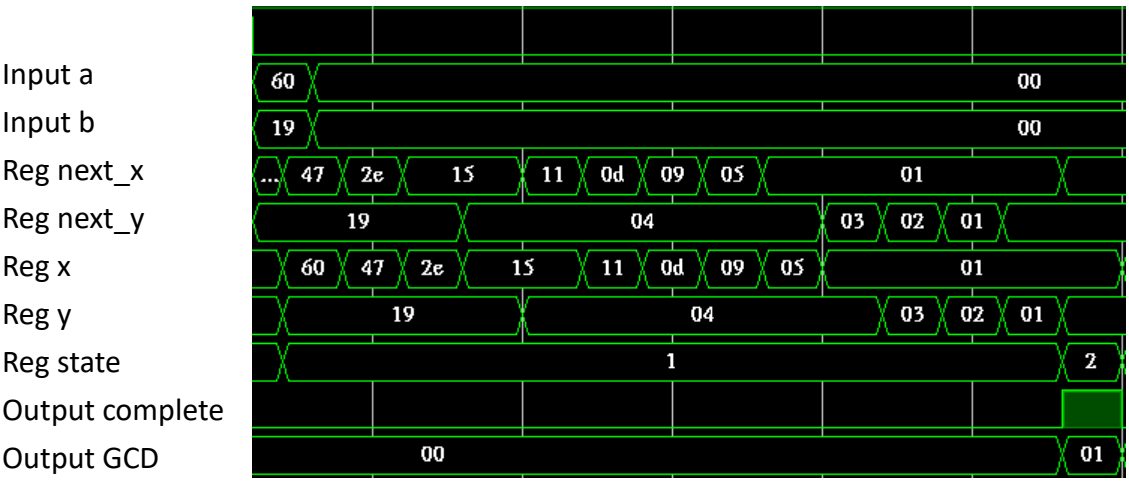
完整的邏輯電路圖如下



Verification

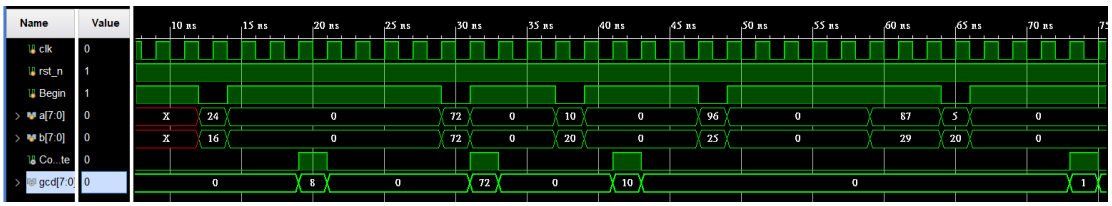
Testbench 設計方式

在跑 simulation 的時候，將 DFF 裡的值(state, x, y)都 output 出來看，觀察在每一個步驟所存的值是否正確，以下為模擬結果圖。



以上 testbench 顯示在 cal 階段都能正常相減，並在運算完之後 state 跳到 finish，將 GCD 結果 output，complete 設為 1。

接著手動輸入幾筆測資及實體電路可能會發生的情況，觀察以下結果是否正確



第一筆為普通的一組數字，結果正確。
第一筆為相同的一組數字，結果正確。
第一筆為互相為倍數的一組數字，結果正確。
第一筆為互質的一組數字，結果正確。
此外，也驗證了在 wait 階段 Begin != 0 的時候，不會運算，且在運算階段時，無論 input 為何，值都不會被寫入，也不會干擾運算進行。

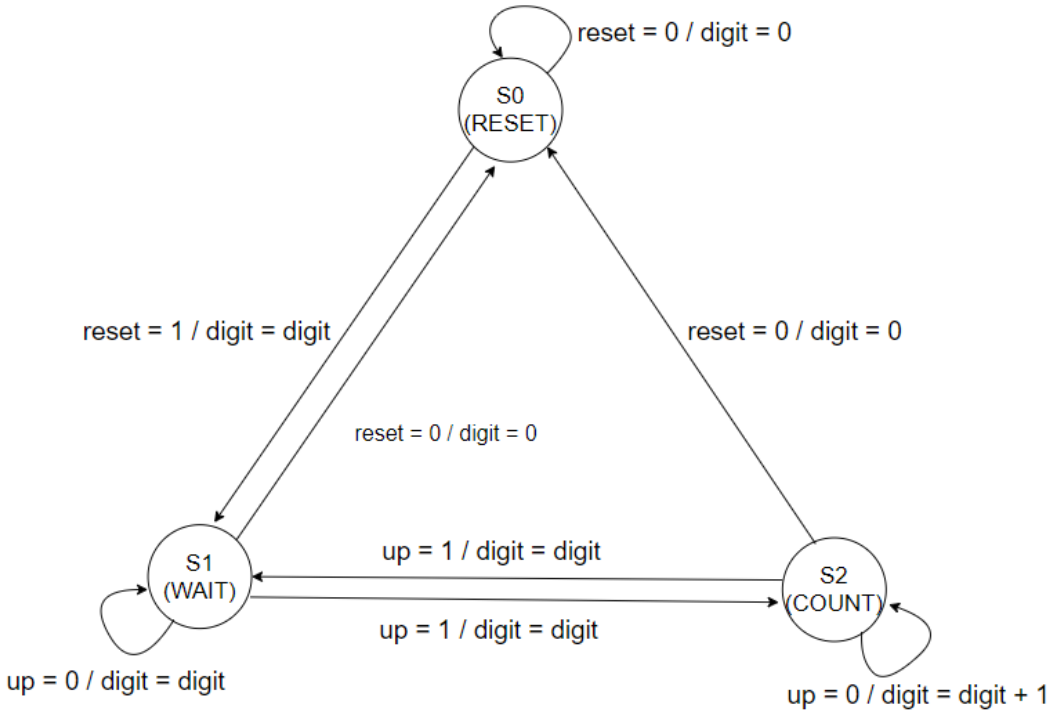
綜合以上結論，可以證明此邏輯電路設計為正確的。

FPGA demonstration

(一)電路設計

首先，我先應題目要求，先畫出此題的 State Transition Diagram，如下：

State：S0 -> RESET,　S1 -> WAIT,　S2 -> COUNT；
Input：reset(right_button), up(up_button)；
Output：digit(displaying on seven segments)；



State Transition Diagram

接著，下方為根據以上的 State Transition Diagram 詳細的設計過程：
(我將七段顯示器上的要顯示的 4 個數字拆成 3 個 output，分別是
[4-1:0] digit0(display digit0), [6-1:0] digit12(display digit1&digit2),
[4-1:0] digit3(display digit3)。

State：S0 -> RESET,　S1 -> WAIT,　S2 -> COUNT；
首先，我需要四個 D-FlipFlop 分別存取 State(分別包含 RESET, WAIT 以及 COUNT),
[4-1:0] digit0, [6-1:0] digit12 以及[4-1:0] digit3。先從 State 的 DFF 開始講起：

1. 如果現在的 State 是 S0(RESET)，下一個 State 則變成 S1；
2. 如果現在的 State 是 S1(WAIT)，需要一個 2_to_1_Mux(控制訊號為 up_button, 如果 up_button = 1, 下一個 State 則變成 S2；如果 up_button = 0,則維持原本的 State)；
3. 如果現在的 State 是 S2(COUNT)，需要一個 2_to_1_Mux(控制訊號為 up_button, 如果 up_button = 1, 下一個 State 則變成 S1；如果 up_button = 0, 則維持原本的 State)；

4. 以上 1. 2. 3.的結果會接到一個 4_to_1_Mux(控制訊號為原本的 State, 如果原本的 State = S0 則輸出 1.的結果；如果原本的 State = S2 則輸出 2.的結果；如果原本的 State = S3 則輸出 3.的結果；default 為結果為 S0) 由下方 State 的 DFF 電路圖可以對照以上所述。

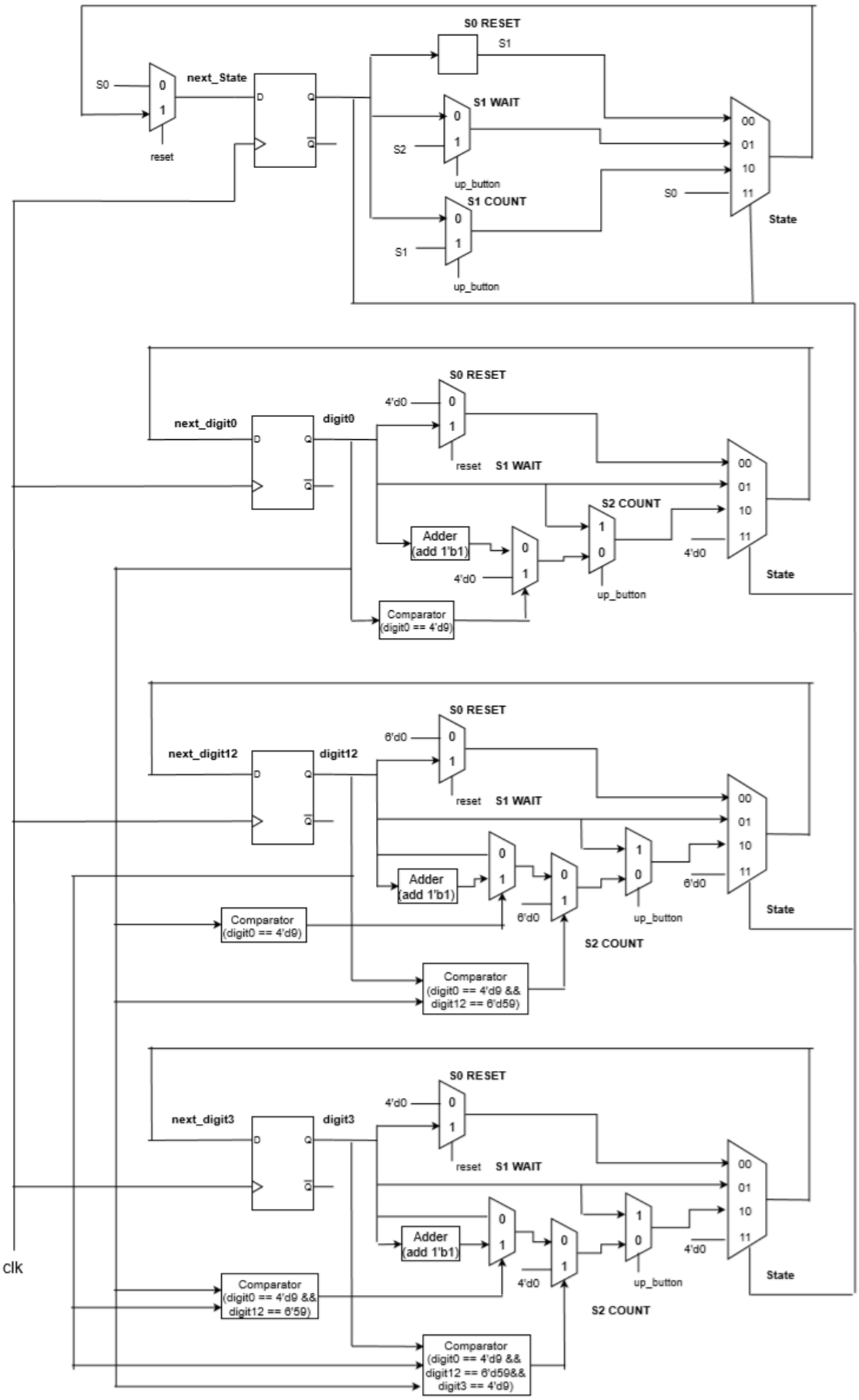
接著，舉 digit12 為例(digit0 以及 digit3 的電路雷同)：

1. 如果現在的 State 是 S0(RESET), 需要一個 2_to_1_Mux(控制訊號為 reset, 如果 reset = 0, digit12 = 6'd0；如果 reset = 1, digit12 維持不變)；
2. 如果現在的 State 是 S1(WAIT)，digit12 維持不變；
3. 如果現在的 State 是 S2(COUNT)，首先前面需要一個 2_to_1_Mux 選擇輸出是否要進位(這時需要一個 Comparator 當作 Mux 的控制訊號，判斷 digit0 是否為 4'd9, 如果 digit0 == 4'd9, 則 digit12 進位加 1；如果 digit0 != 4'd9, 則 digit12 維持不變)。判斷完是否需要進位後，此時需要再判斷 digit12 是否會溢位，所以需要再接一個 2_to_1_Mux(這時需要另一個 Comparator 當作 Mux 的控制訊號，判斷(digit0 == 4'd9 && digit12 == 6'd59), 如果控制訊號為 1, 則 digit12 = 6d'0；如果控制訊號為 0, 則 digit12 維持不變)。最後後面還需要一個 2_to_1_Mux(控制訊號為 up_button，如果 up_button = 1(轉換成 WAIT State), digit12 會變成原本的 digit12；如果 up_button = 0(持續為 COUNT State), digit12 為接完前面兩個 2_to_1_Mux 的輸出)；
4. 以上 1. 2. 3.的結果會接到一個 4_to_1_Mux(控制訊號為原本的 State(在 State 的 DFF), 如果原本的 State = S0 則輸出 1.的結果；如果原本的 State = S2 則輸出 2.的結果；如果原本的 State = S3 則輸出 3.的結果；default 為結果為 6'd0) 由下方 digit12 的 DFF 電路圖可以對照以上所述。

#digit0 以及 digit3 跟 digit12 的差別只有在上述的 3.的進位條件以及溢位條件：

1. digit0 是每次都會進位，而當 digit == 4'd9 時會溢位；
2. digit3 是當(digit0 == 4'd9 && digit12 == 6'd59) == 1 時才會進位，而當(digit0 == 4'd9 && digit12 == 6'd59 && digit3 == 4'd9) == 1 時會溢位。

下方的電路圖可實現以上我的設計理念：


S0 RESET
S1
S0
next_State
S1 WAIT
S2
up_button
S1 COUNT
S1
up_button
S0
State
reset
next_digit0
digit0
4'd0
S0 RESET
reset
S1 WAIT
S2 COUNT
Adder
(add 1'b1)
4'd0
up_button
Comparator
(digit0 == 4'd9)
State
next_digit12
digit12
6'd0
S0 RESET
reset
S1 WAIT
Adder
(add 1'b1)
6'd0
up_button
Comparator
(digit0 == 4'd9)
S2 COUNT
Comparator
(digit0 == 4'd9 &&
digit12 == 6'd59)
State
next_digit3
digit3
4'd0
S0 RESET
reset
S1 WAIT
Adder
(add 1'b1)
4'd0
up_button
Comparator
(digit0 == 4'd9 &&
digit12 == 6'd59)
S2 COUNT
Comparator
(digit0 == 4'd9 &&
digit12 == 6'd59&&
digit3 == 4'd9)
State
clk

(二)FPGA Demostration

接下來我們要將 verilog code 跑在板子上面，此分為兩個重要的環節：

(a)顯示：

1.decoder :為了將數字轉換為阿拉伯數字圖形，必須設計一個 seven_seg decoder，當要表示一個數字時，會有對應的一組控制七段顯示器的訊號輸出。

2.七段顯示器控制 : 為了讓每個七段顯示器輪流亮燈，用 2-bit 的 sctl_clk 訊號接一個 digit_display decoder 控制(4-bit 的 output(digit_display))，一個時間只會有一個七段顯示器亮(ex:當 sctl_clk = 2'b00, digit_display = 4'b1110(只亮 digit0,其他皆暗掉))。

(b)訊號處理：

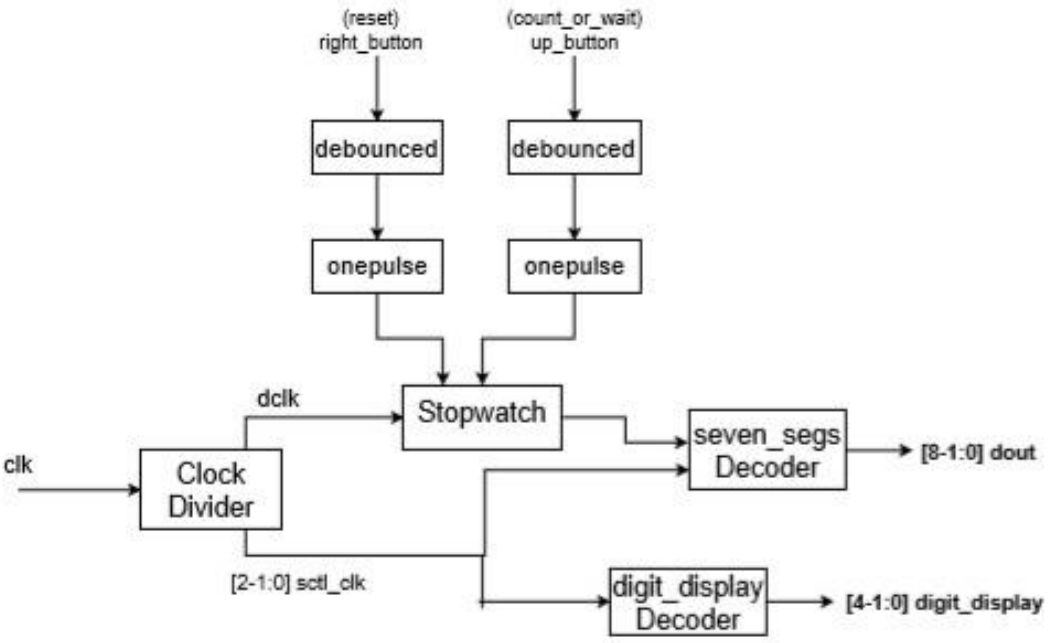
1.Clock Divider : 首先我們要將石英震盪器的頻率除頻至人眼可已看得到的頻率，並且 digit0 的週期是相當於 0.1 秒。經過層層測試，我得到以下結果：先做一個 24-bit 的加法器(此加法器當 num[23] == 1'b1 && num[20] == 1'b1 && num[19] == 1'b1 時，加法器歸零(num <= 24'd0))，最高位的那一項輸出(num[24])作為 Stopwatch 的頻率，此外我們也輸出了第 15 和第 14 個 bit 作為輪流顯示七段顯示器的控制訊號(sctl_clk = num[15:14])。

2.debounce : 由於用按壓按鈕的初期，按鈕會震盪而產生雜訊，為了過濾訊號，接了若干組 DFF，當每個 DFF 的值皆為 1 時，才輸出 1。

3.onepulse : 在 reset, up_button 作用的時候，我們只需要效果持續一個週期，因此在 debounce 完之後，會接一個反向器做 AND，讓 reset, up_button 的值為 1 的時間僅有 1 個週期。

(c)電路檢測方式：看 FPGA 板上所顯示的情形是否符合 specification。

以下為 FPGA 的 Block Diagram：



心得

很高興這次的邏設實驗打得比較快，沒有上次痛苦的 debug 過程，可能是 verilog 能力提升了，又或是這次的題目比較平易近人一點點，希望以後邏設實驗打起來都可以這麼快樂。

這次畫圖的時間同樣很長，希望可以趕快熟悉這個軟體的使用，雖然畫圖是真的很煩，不過在畫完圖對於整個邏輯電路的理解更加清晰。

(107060015 陳弘軽)

這次的 Lab 是 Finite State Machine，我覺得比前面幾次 Lab 都還有趣的多，因為設計的元素又多了些，而且我 debug 的技巧又提升了不少，因此總共花費的時間沒有像上次的 Lab3 這麼多，加上這次 Lab4 我比較早開始打，最後還有時間去優化自己的 code，還幫助了其他組的組員，這一次的 Lab4 算是最開心的一次，雖然前幾次以及期中考的分數不是很理想，但是我會堅持下去的！

(107060011 涂皓鈞)

小組分工：

涂皓鈞－負責 FPGA 題 verilog 以及對應題號的結報內容 + 心得
陳弘軽－負責 Question1, 2 的 verilog 以及對應題號的結報內容 + 心得